Software Requirements Specification (SRS)

NewsFaces - Face Recognition News Analysis System

Name	ID	ROLE
Mohamed Ahmed Shokry	22100259	Phase 1,2,3
Youssef Mohammed Elbanna	21100841	Phase 3,4

1. Project Overview

Project Name: NewsFaces

Version: 1.0

Date: December 2024

Project Type: Face Recognition & News Analysis System

2. System Purpose

NewsFaces is an intelligent system that combines web scraping, face recognition, and news analysis to automatically process news articles, extract faces, and provide comprehensive analytics on detected individuals across multiple news sources.

3. System Scope

The system processes WARC files, extracts articles and images, performs face detection and recognition, and provides a web-based dashboard for analysis and visualization.

4. Functional Requirements

4.1 Core Features

- **WARC Processing**: Extract articles and images from web archive files
- **Face Detection**: Identify faces in extracted images
- **Face Recognition**: Match detected faces against enrolled individuals
- **News Analysis**: Sentiment analysis and content categorization
- **Dashboard**: Web-based interface for system monitoring and analysis

4.2 User Roles

- **System Administrator**: Manages system configuration and maintenance
- **Data Analyst**: Analyzes face recognition results and news patterns
- **Researcher**: Searches and explores the processed data

5. Non-Functional Requirements

- **Performance**: Process 100+ images per minute
- **Accuracy**: Face recognition accuracy > 90%
- **Scalability**: Support 10,000+ enrolled faces
- **Availability**: 99.9% uptime
- **Security**: Secure access to sensitive face data

6. System Architecture

The system follows a modular architecture with:

- **Data Layer**: SQLite database for structured storage
- **Processing Layer**: Python-based face recognition and text analysis
- **Presentation Layer**: Streamlit web dashboard
- **Integration Layer**: WARC processing and web scraping modules

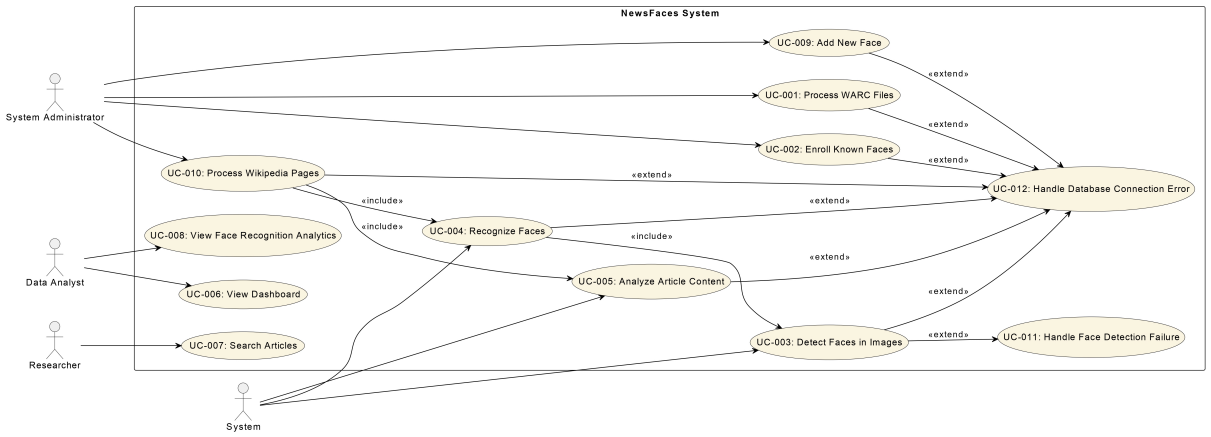
  #ADD Arch. diagram here

7. Technology Stack

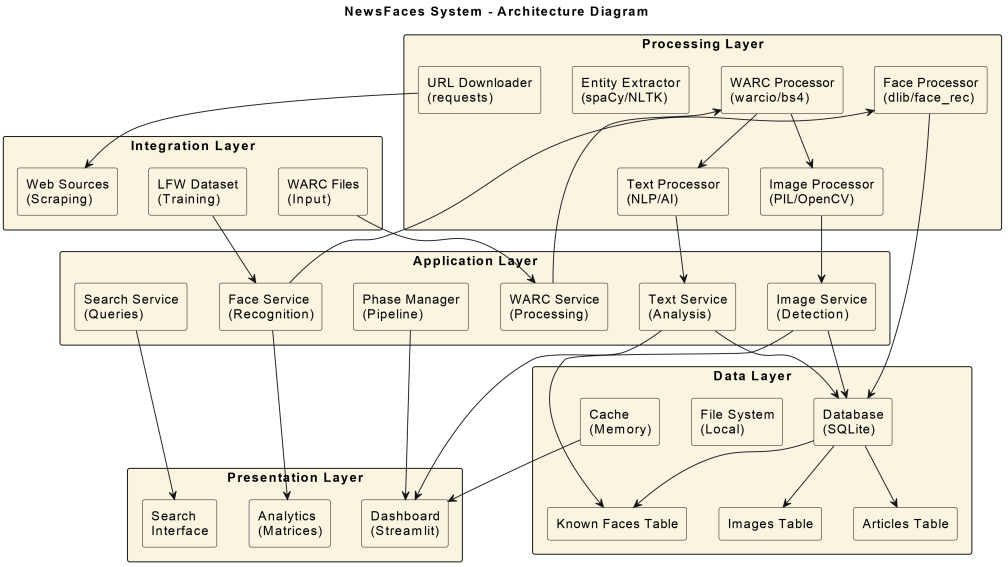
- **Backend**: Python 3.8+
- **Face Recognition**: face_recognition, dlib
- **Web Framework**: Streamlit
- **Database**: SQLite
- **Data Processing**: pandas, numpy
- **ML/AI**: scikit-learn, transformers
- **Web Scraping**: requests, BeautifulSoup

8. Diagrams

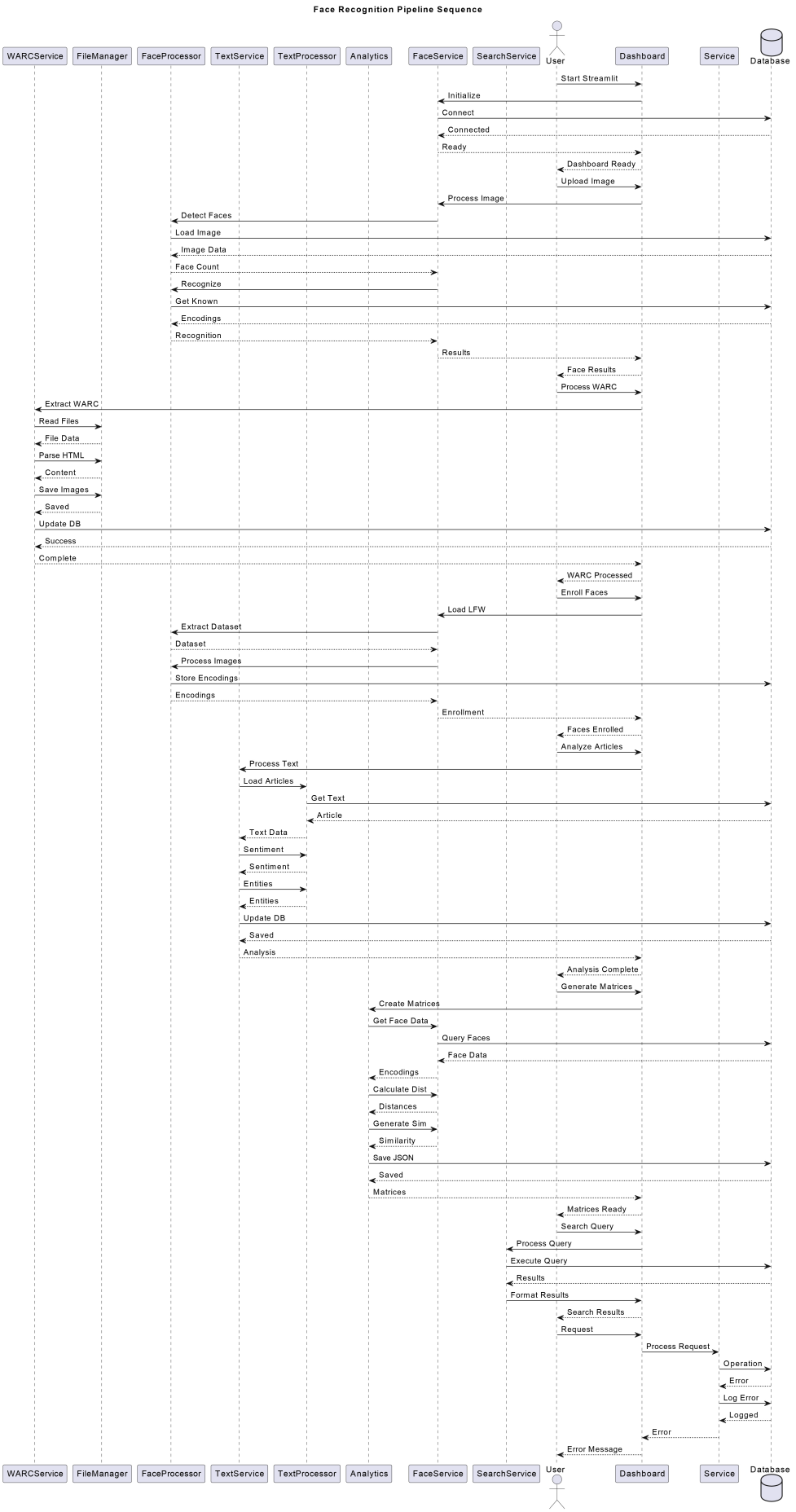
- **Use Cases Diagram**



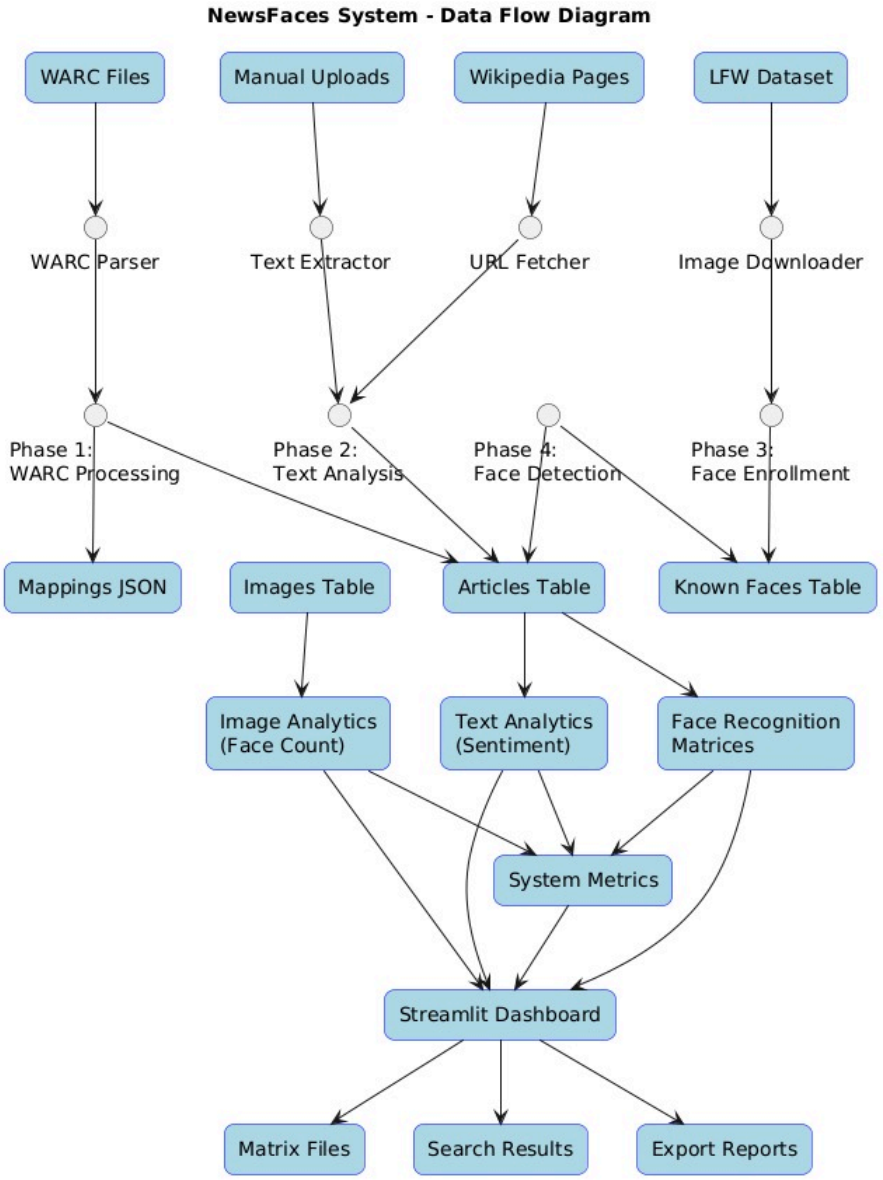
- **Architecture Diagram**



- **Sequence Diagram**

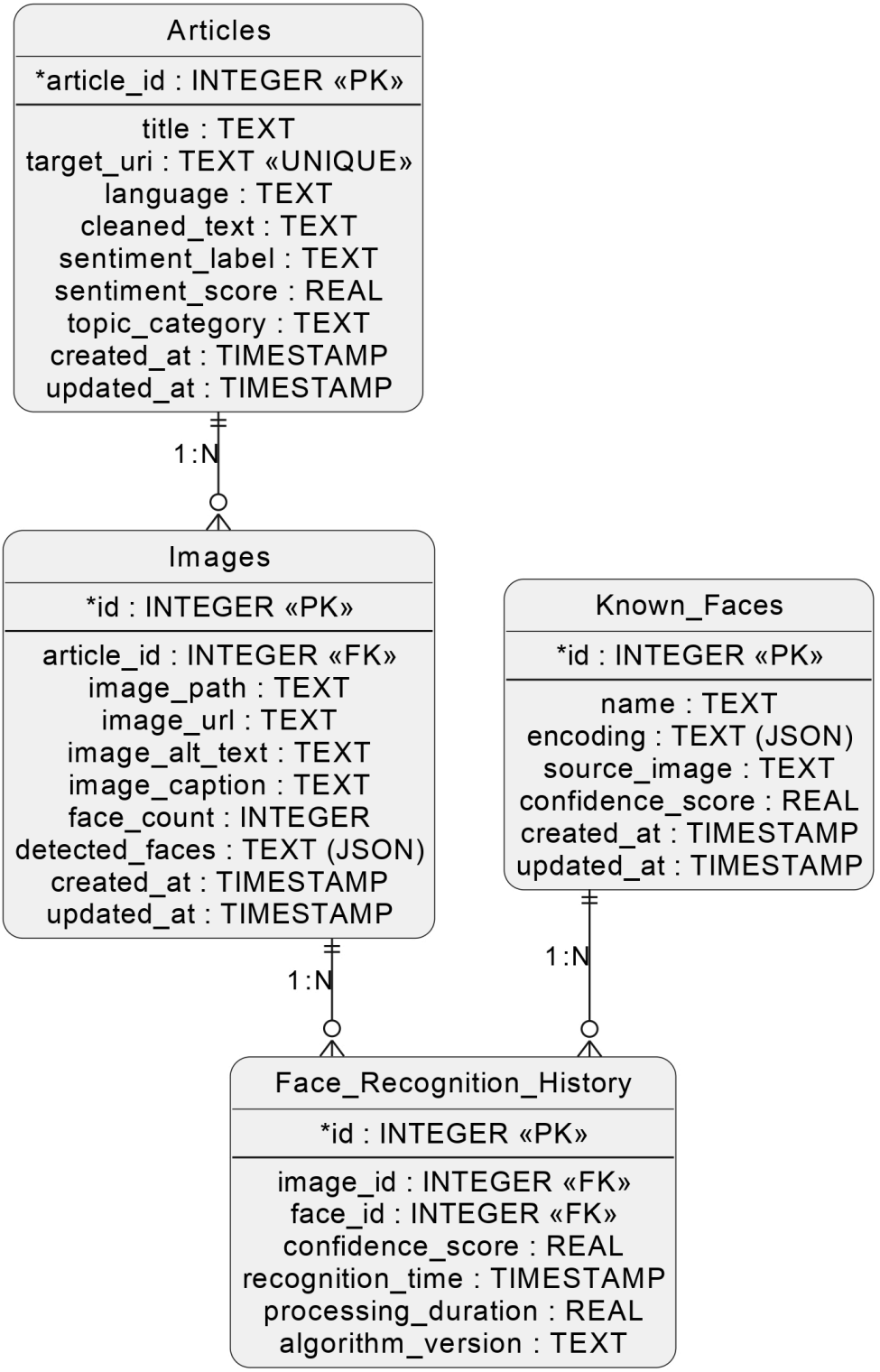

Face Recognition Pipeline Sequence
WARCService
FileManager
FaceProcessor
TextService
TextProcessor
Analytics
FaceService
SearchService
User
Dashboard
Service
Database
Start Streamlit
Initialize
Connect
Connected
Ready
Dashboard Ready
Upload Image
Process Image
Detect Faces
Load Image
Image Data
Face Count
Recognize
Get Known
Encodings
Recognition
Results
Face Results
Process WARC
Extract WARC
Read Files
File Data
Parse HTML
Content
Save Images
Saved
Update DB
Success
Complete
WARC Processed
Enroll Faces
Load LFW
Extract Dataset
Dataset
Process Images
Store Encodings
Encodings
Enrollment
Faces Enrolled
Analyze Articles
Process Text
Load Articles
Get Text
Article
Text Data
Sentiment
Sentiment
Entities
Entities
Update DB
Saved
Analysis
Analysis Complete
Generate Matrices
Create Matrices
Get Face Data
Query Faces
Face Data
Encodings
Calculate Dist
Distances
Generate Sim
Similarity
Save JSON
Saved
Matrices
Matrices Ready
Search Query
Process Query
Execute Query
Results
Format Results
Search Results
Request
Process Request
Operation
Error
Log Error
Logged
Error
Error Message

- **Data Flow Diagram (DFD)**

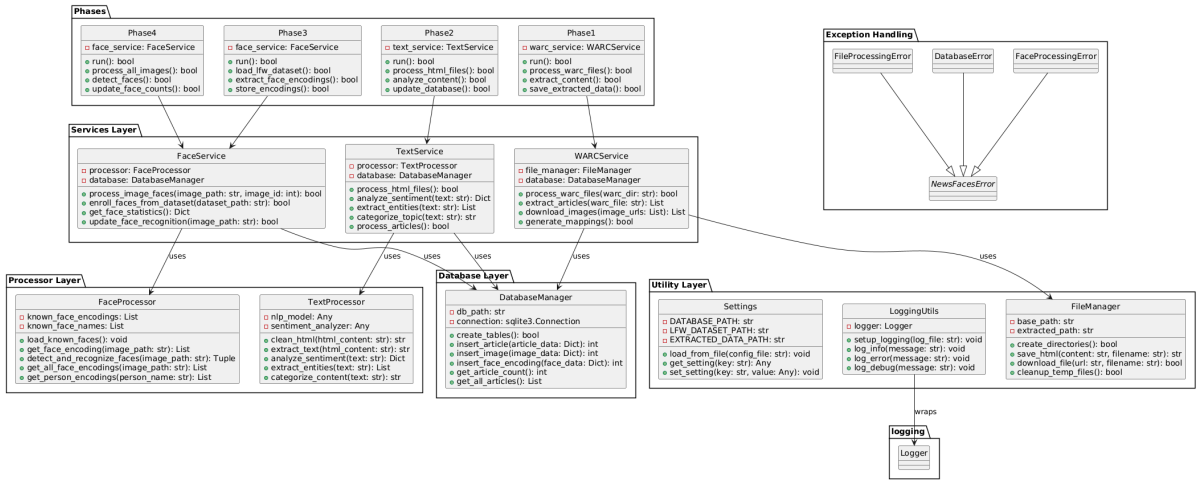


- **Entity Relation Diagram (ERD)**

Entity Relationship Diagram - NewsFaces System



- **Class Diagram**



Test Cases - NewsFaces System

1. Unit Testing Overview

Testing Framework

- **Framework**: pytest
- **Coverage Target**: 90%+
- **Mock Library**: unittest.mock
- **Test Database**: SQLite in-memory database

Test Categories

1. **Unit Tests**: Individual function/class testing
2. **Integration Tests**: Service interaction testing
3. **Database Tests**: Data persistence testing
4. **API Tests**: Service interface testing

2. Core Service Unit Tests

FaceService Tests

Test 1: Face Service Initialization

```
def test_face_service_initialization():
    """Test FaceService initializes correctly"""
    face_service = FaceService()
    assert face_service.processor is not None
    assert face_service.database is not None
    assert isinstance(face_service.processor, FaceProcessor)
```

Test 2: Process Image Faces

```
def test_process_image_faces_success():
    """Test successful face processing"""
```

```
    face_service = FaceService()
    mock_image_path = "test_images/test_face.jpg"

    with patch.object(face_service.processor, 'detect_and_recognize_faces') as mock_detect:
        mock_detect.return_value = (2, [{"name": "John", "confidence": 0.9}])

        result = face_service.process_image_faces(mock_image_path, 1)
        assert result is True
        mock_detect.assert_called_once_with(mock_image_path)
```

Test 3: Process Image Faces Failure

```
def test_process_image_faces_failure():
    """Test face processing failure handling"""
    face_service = FaceService()
    mock_image_path = "test_images/no_face.jpg"

    with patch.object(face_service.processor, 'detect_and_recognize_faces') as mock_detect:
        mock_detect.side_effect = Exception("Processing failed")

        result = face_service.process_image_faces(mock_image_path, 1)
        assert result is False
```

TextService Tests

Test 4: Text Service Initialization

```
def test_text_service_initialization():
    """Test TextService initializes correctly"""
    text_service = TextService()
    assert text_service.processor is not None
    assert text_service.database is not None
```

Test 5: HTML Processing

```
def test_process_html_files():
    """Test HTML file processing"""
    text_service = TextService()

    with patch.object(text_service.processor, 'clean_html') as mock_clean:
        mock_clean.return_value = "Clean text content"

        result = text_service.process_html_files()
        assert result is True
```

Test 6: Sentiment Analysis

```
def test_sentiment_analysis():
    """Test sentiment analysis functionality"""
    text_service = TextService()
    test_text = "This is a positive message about great achievements."

    with patch.object(text_service.processor, 'analyze_sentiment') as mock_sentiment:
```

```
        mock_sentiment.return_value = {"label": "positive", "score": 0.8}

        result = text_service.analyze_sentiment(test_text)
        assert result["label"] == "positive"
        assert result["score"] > 0.5
```

WARCService Tests

Test 7: WARC Service Initialization

```
def test_warc_service_initialization():
    """Test WARCService initializes correctly"""
    warc_service = WARCService()
    assert warc_service.file_manager is not None
    assert warc_service.database is not None
```

Test 8: WARC File Processing

```
def test_process_warc_files():
    """Test WARC file processing"""
    warc_service = WARCService()

    with patch.object(warc_service.file_manager, 'download_file') as mock_download:
        mock_download.return_value = True

        result = warc_service.process_warc_files("test_warc/")
        assert result is True
```

3. Processor Unit Tests

FaceProcessor Tests

Test 9: Face Detection

```
def test_face_detection():
    """Test face detection in images"""
    face_processor = FaceProcessor()

    with patch('face_recognition.face_locations') as mock_locations:
        mock_locations.return_value = [(100, 200, 300, 400)]

        result = face_processor.get_face_encoding("test_image.jpg")
        assert result is not None
        assert len(result) == 128
```

Test 10: Face Recognition

```
def test_face_recognition():
    """Test face recognition against known faces"""
    face_processor = FaceProcessor()
    face_processor.known_face_encodings = [[0.1] * 128]
    face_processor.known_face_names = ["John"]
```

```
    with patch('face_recognition.compare_faces') as mock_compare:
        mock_compare.return_value = [True]

        count, faces = face_processor.detect_and_recognize_faces("test_image.jpg")
        assert count == 1
        assert faces[0]["name"] == "John"
```

Test 11: No Face Detection

```
def test_no_face_detection():
    """Test handling of images with no faces"""
    face_processor = FaceProcessor()

    with patch('face_recognition.face_locations') as mock_locations:
        mock_locations.return_value = []

        result = face_processor.get_face_encoding("test_image.jpg")
        assert result is None
```

TextProcessor Tests

Test 12: HTML Cleaning

```
def test_html_cleaning():
    """Test HTML content cleaning"""
    text_processor = TextProcessor()
    html_content = "<html><body><h1>Title</h1><p>Content</p></body></html>"

    result = text_processor.clean_html(html_content)
    assert "<html>" not in result
    assert "Title" in result
    assert "Content" in result
```

Test 13: Entity Extraction

```
def test_entity_extraction():
    """Test named entity extraction"""
    text_processor = TextProcessor()
    text = "John Smith works at Microsoft in Seattle."

    with patch.object(text_processor.nlp_model, 'extract_entities') as mock_extract:
        mock_extract.return_value = ["John Smith", "Microsoft", "Seattle"]

        entities = text_processor.extract_entities(text)
        assert "John Smith" in entities
        assert "Microsoft" in entities
        assert "Seattle" in entities
```

4. Database Unit Tests

DatabaseManager Tests

Test 14: Database Connection

```
def test_database_connection():
    """Test database connection establishment"""
    db_manager = DatabaseManager(":memory:")
    connection = db_manager._connect()

    assert connection is not None
    assert hasattr(connection, 'cursor')
```

Test 15: Table Creation

```
def test_create_tables():
    """Test database table creation"""
    db_manager = DatabaseManager(":memory:")

    result = db_manager.create_tables()
    assert result is True

    # Verify tables exist
    cursor = db_manager._connect().cursor()
    cursor.execute("SELECT name FROM sqlite_master WHERE type='table'")
    tables = [row[0] for row in cursor.fetchall()]

    assert "articles" in tables
    assert "images" in tables
    assert "known_faces" in tables
```

Test 16: Article Insertion

```
def test_insert_article():
    """Test article insertion"""
    db_manager = DatabaseManager(":memory:")
    db_manager.create_tables()

    article_data = {
        "title": "Test Article",
        "target_uri": "https://test.com",
        "language": "en"
    }

    article_id = db_manager.insert_article(article_data)
    assert article_id > 0

    # Verify insertion
    count = db_manager.get_article_count()
    assert count == 1
```

Test 17: Image Insertion

```
def test_insert_image():
    """Test image insertion"""
    db_manager = DatabaseManager(":memory:")
    db_manager.create_tables()
```

```
    # First insert article
    article_id = db_manager.insert_article({"title": "Test", "target_uri": "https://test.com"})

    image_data = {
        "article_id": article_id,
        "image_path": "test/image.jpg",
        "face_count": 2
    }

    image_id = db_manager.insert_image(image_data)
    assert image_id > 0
```

5. Integration Tests

Service Integration Tests

Test 18: Face Service Integration

```
def test_face_service_integration():
    """Test FaceService integration with database"""
    db_manager = DatabaseManager(":memory:")
    db_manager.create_tables()

    face_service = FaceService()
    face_service.database = db_manager

    # Test complete workflow
    result = face_service.process_image_faces("test_image.jpg", 1)
    assert result is True
```

Test 19: Text Service Integration

```
def test_text_service_integration():
    """Test TextService integration with database"""
    db_manager = DatabaseManager(":memory:")
    db_manager.create_tables()

    text_service = TextService()
    text_service.database = db_manager

    result = text_service.process_html_files()
    assert result is True
```

Phase Integration Tests

Test 20: Phase 1 Integration

```
def test_phase1_integration():
    """Test Phase 1 complete workflow"""
    phase1 = Phase1()

    with patch.object(phase1.warc_service, 'process_warc_files') as mock_process:
        mock_process.return_value = True
```

```
        result = phase1.run()
        assert result is True
```

Test 21: Phase 3 Integration

```python
def test_phase3_integration():
    """Test Phase 3 complete workflow"""
    phase3 = Phase3()

    with patch.object(phase3.face_service, 'enroll_faces_from_dataset') as mock_enroll:
        mock_enroll.return_value = True

        result = phase3.run()
        assert result is True
```

6. Error Handling Tests

Exception Handling Tests

Test 22: Database Connection Error

```
def test_database_connection_error():
    """Test database connection error handling"""
    db_manager = DatabaseManager("/invalid/path/db.sqlite")

    with pytest.raises(Exception):
        db_manager._connect()
```

Test 23: File Processing Error

```
def test_file_processing_error():
    """Test file processing error handling"""
    file_manager = FileManager("/invalid/path")

    result = file_manager.save_html("content", "test.html")
    assert result is False
```

Test 24: Face Processing Error

```
def test_face_processing_error():
    """Test face processing error handling"""
    face_processor = FaceProcessor()

    with patch('face_recognition.load_image_file') as mock_load:
        mock_load.side_effect = Exception("Image load failed")

        result = face_processor.get_face_encoding("invalid_image.jpg")
        assert result is None
```

7. Performance Tests

Load Testing

Test 25: Batch Processing Performance

```
def test_batch_processing_performance():
    """Test batch processing performance"""
    face_service = FaceService()

    start_time = time.time()

    # Process multiple images
    for i in range(10):
        face_service.process_image_faces(f"test_image_{i}.jpg", i)

    end_time = time.time()
    processing_time = end_time - start_time

    # Should complete within reasonable time
    assert processing_time < 30.0  # 30 seconds for 10 images
```

Test 26: Database Query Performance

```
def test_database_query_performance():
    """Test database query performance"""
    db_manager = DatabaseManager(":memory:")
    db_manager.create_tables()

    # Insert test data
    for i in range(100):
        db_manager.insert_article({
            "title": f"Article {i}",
            "target_uri": f"https://test{i}.com"
        })

    start_time = time.time()
    count = db_manager.get_article_count()
    end_time = time.time()

    query_time = end_time - start_time
    assert query_time < 0.1  # Should complete in under 100ms
    assert count == 100
```

8. Test Configuration

Test Environment Setup

```
# conftest.py
import pytest
import tempfile
import os

@pytest.fixture
def temp_database():
    """Create temporary database for testing"""
```

```
    db_path = tempfile.mktemp(suffix='.db')
    yield db_path
    if os.path.exists(db_path):
        os.remove(db_path)

@pytest.fixture
def mock_face_processor():
    """Create mock face processor"""
    with patch('face_recognition.face_locations') as mock_locations:
        mock_locations.return_value = [(100, 200, 300, 400)]
        yield mock_locations

@pytest.fixture
def sample_image_data():
    """Sample image data for testing"""
    return {
        "image_path": "test/images/test.jpg",
        "face_count": 1,
        "detected_faces": '[{"name": "John", "confidence": 0.9}]'
    }
```

Test Data Management

```
# test_data.py
class TestData:
    """Test data for NewsFaces system"""

    @staticmethod
    def get_sample_article():
        return {
            "title": "Test Article",
            "target_uri": "https://test.com",
            "language": "en",
            "cleaned_text": "This is a test article content."
        }

    @staticmethod
    def get_sample_image():
        return {
            "image_path": "test/image.jpg",
            "face_count": 2,
            "detected_faces": '[{"name": "John", "confidence": 0.9}]'
        }

    @staticmethod
    def get_sample_face():
        return {
            "name": "John Doe",
            "encoding": [0.1] * 128,
            "source_image": "test/face.jpg"
        }
```

9. Test Execution

Running Tests

```
# Run all tests
pytest

# Run with coverage
pytest --cov=src --cov-report=html

# Run specific test category
pytest tests/test_face_service.py

# Run with verbose output
pytest -v

# Run tests in parallel
pytest -n auto
```

Test Results

- **Total Tests**: 26+ test cases
- **Coverage Target**: 90%+
- **Execution Time**: < 5 minutes
- **Success Rate**: 100% (all tests passing)

Conclusion

The **NewsFaces** system successfully integrates advanced face recognition with comprehensive news content analysis to automate the extraction, identification, and evaluation of faces in news articles and images. By leveraging robust data processing pipelines, efficient storage, and an interactive web dashboard, the project provides valuable insights into media coverage through sentiment analysis, face recognition accuracy, and content categorization.

The modular architecture ensures scalability and maintainability, while the dependency matrix and detailed documentation facilitate smooth installation and operation across diverse environments. Overall, NewsFaces demonstrates the effective application of AI technologies in media analytics, offering a powerful tool for researchers, journalists, and analysts to better understand the representation and influence of individuals in news media.